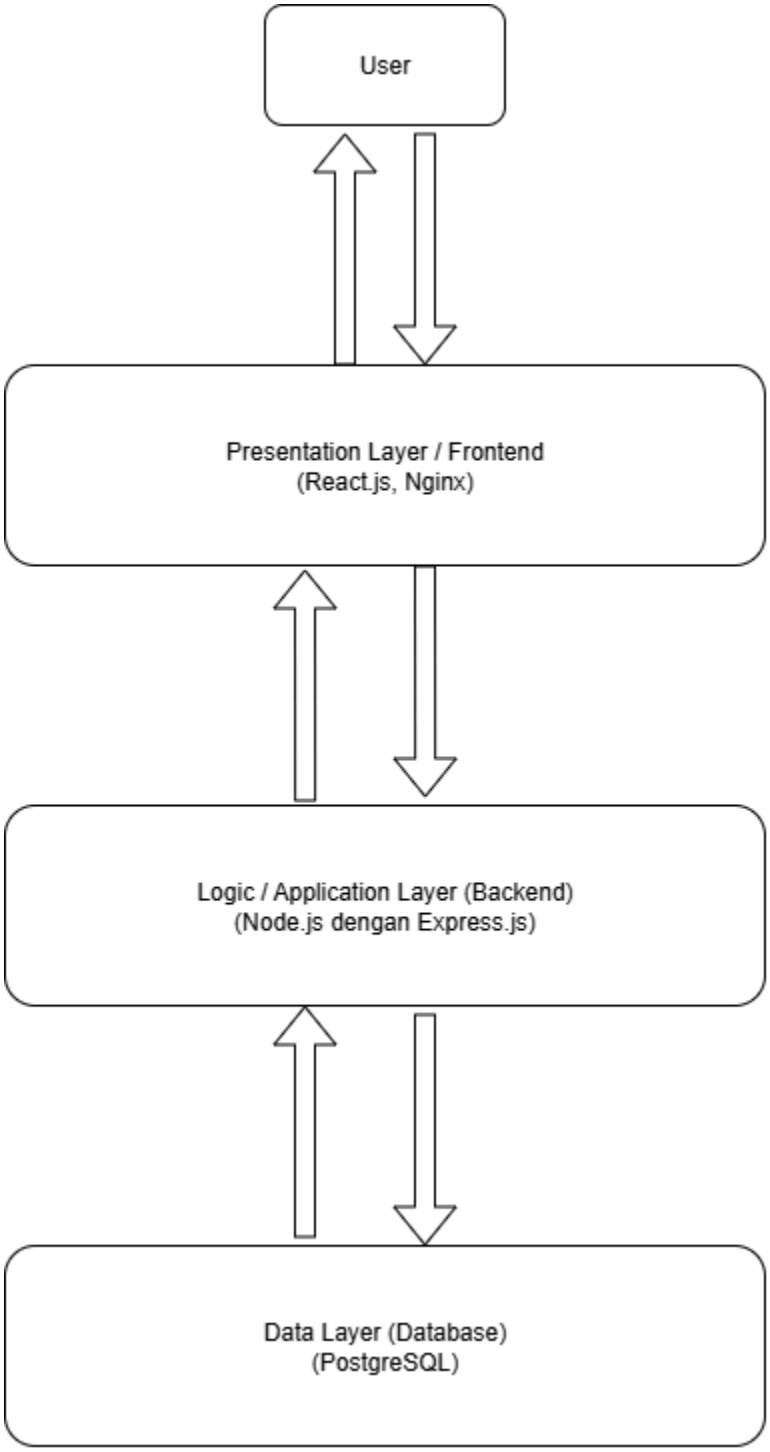
Pendefinisian Web Application

Pada tugas ini, saya merancang sebuah To-Do List berbasis web application sederhana yang memiliki fungsi untuk menambahkan list, menghapus list, dan menandai list. Aplikasi ini dikembangkan menggunakan platform pengembangan modern dengan kontainer. Pada perancangannya saya menggunakan Arsitektur 3-Layer sebagai berikut :



1. Presentation Layer (Frontend)

- Peran (Wajah Aplikasi): Lapisan ini adalah satu-satunya bagian yang berinteraksi langsung dengan pengguna. Tugas utamanya adalah menampilkan antarmuka pengguna (UI), menyajikan data yang diterima dari backend, dan menangkap input dari pengguna (seperti klik tombol atau pengisian form). Lapisan ini tidak berisi logika bisnis yang kompleks.
- Teknologi yang Digunakan:
    - React.js: Untuk membangun komponen UI yang interaktif dan dinamis.
    - Nginx: Sebagai web server ringan yang bertugas menyajikan file-file statis (HTML, CSS, JavaScript) hasil *build* dari React ke browser pengguna.

2. Logic/Application Layer (Backend)

- Peran (Otak Aplikasi): Lapisan ini adalah pusat dari semua proses bisnis. Ia menerima permintaan dari Presentation Layer (misalnya, "tambahkan tugas baru"), memvalidasi data, menjalankan logika yang sesuai, dan berkomunikasi dengan Data Layer untuk menyimpan atau mengambil informasi. Lapisan ini diekspos sebagai API (Application Programming Interface) yang digunakan oleh frontend.
- Teknologi yang Digunakan:
    - Node.js dengan Express.js: Untuk membuat server API yang efisien, menangani *routing* (misal: /api/tasks), dan mengelola permintaan HTTP.

3. Data Layer (Database)

- Peran (Penyimpanan Data): Lapisan ini bertindak sebagai "lemari arsip" aplikasi. Tanggung jawab utamanya adalah untuk menyimpan, mengelola, dan mengambil data secara persisten dan aman. Lapisan ini tidak peduli dengan logika bisnis atau tampilan; ia hanya fokus pada operasi data (Create, Read, Update, Delete - CRUD).
- Teknologi yang Digunakan:
    - PostgreSQL: Sebagai sistem manajemen database relasional yang andal untuk menyimpan semua data tugas secara terstruktur di dalam tabel.

Tahapan Membangun Aplikasi Web dengan Kontainer

Proses pembangunan aplikasi web *To-Do List* berdasarkan arsitektur tiga tingkat dengan menggunakan kontainer dapat dibagi menjadi tiga tahapan utama: Perencanaan, Implementasi, dan Orkestrasi Lokal.

Tahap 1: Perencanaan dan Desain Arsitektur

1. Mendefinisikan Kebutuhan: Kebutuhan fungsional ditentukan (melihat, menambah, mengubah status, menghapus tugas) untuk menetapkan lingkup proyek.
2. Pemilihan Arsitektur: Arsitektur Tiga Tingkat (Three-Tier Architecture) dipilih.
    - Analisis: Arsitektur ini ideal karena memisahkan secara jelas antara tampilan (Presentation), logika bisnis (Logic), dan penyimpanan data (Data). Pemisahan ini membuat aplikasi lebih mudah dikembangkan, dipelihara, dan diskalakan di kemudian hari.
3. Pemilihan Teknologi per Layer:
    - Presentation (Frontend): React.js dipilih untuk menciptakan antarmuka yang dinamis dan responsif (Single-Page Application).
    - Logic (Backend): Node.js + Express.js dipilih karena performanya yang cepat untuk operasi I/O (cocok untuk API) dan memungkinkan penggunaan JavaScript di seluruh tumpukan teknologi (full-stack).
    - Data (Database): PostgreSQL dipilih karena merupakan sistem database relasional yang andal, gratis, dan sangat cocok untuk menyimpan data terstruktur seperti daftar tugas.
    - Platform Containerization: Docker dipilih sebagai platform utama.
        - Analisis: Docker menyelesaikan masalah klasik "works on my machine". Dengan membungkus setiap *tier* ke dalam kontainer, kita memastikan aplikasi berjalan secara konsisten di lingkungan mana pun (laptop developer, server produksi), sehingga membuktikan bahwa desainnya portabel dan dapat diimplementasikan.

Tahap 2: Implementasi dan Containerization

Pada tahap ini, kode aplikasi ditulis dan dibungkus ke dalam unit-unit portabel. (untuk lebih lengkap, hasil pengembangan kode dapat dilihat di repository github)

1. Pengembangan Kode: Kode untuk setiap *tier* ditulis secara terpisah di dalam foldernya masing-masing (backend/ dan frontend/).
2. Pembuatan Dockerfile: Untuk setiap *tier* aplikasi (backend dan frontend), sebuah Dockerfile dibuat.
    - Fungsi: Dockerfile bertindak sebagai resep atau cetak biru untuk membangun sebuah Docker Image. *Image* ini adalah paket *standalone* yang berisi semua yang dibutuhkan aplikasi untuk berjalan: kode, *runtime* (misal: Node.js), *library*, dan variabel lingkungan. Ini adalah langkah inti dari "menggunakan kontainer".

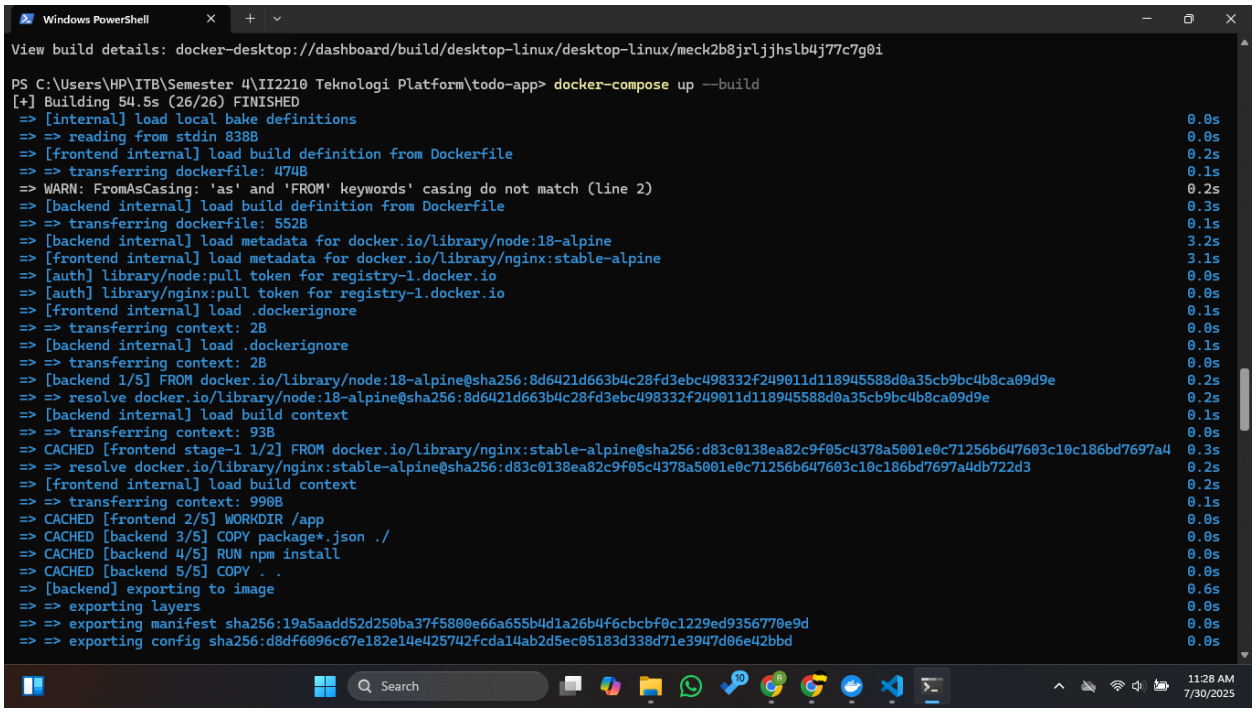
Tahap 3: Orkestrasi Lokal dan Pengujian

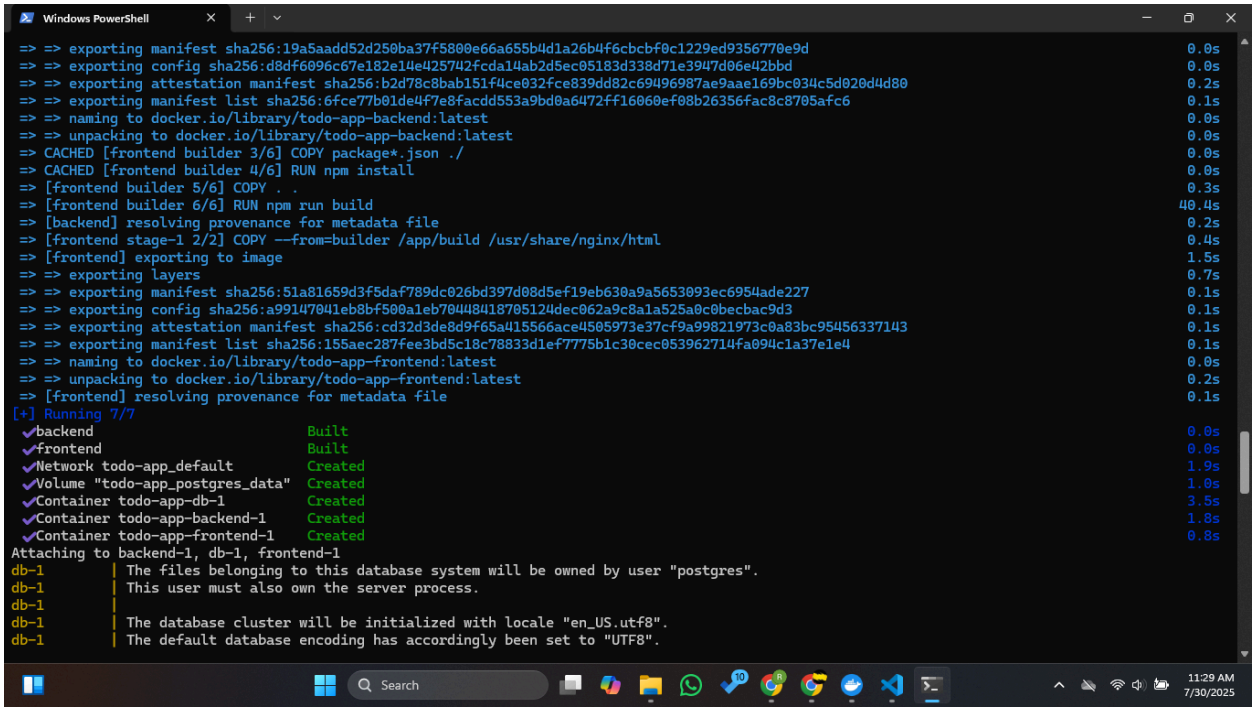
Pada tahap ini, semua kontainer yang terpisah dijalankan secara bersamaan sebagai satu aplikasi yang utuh.

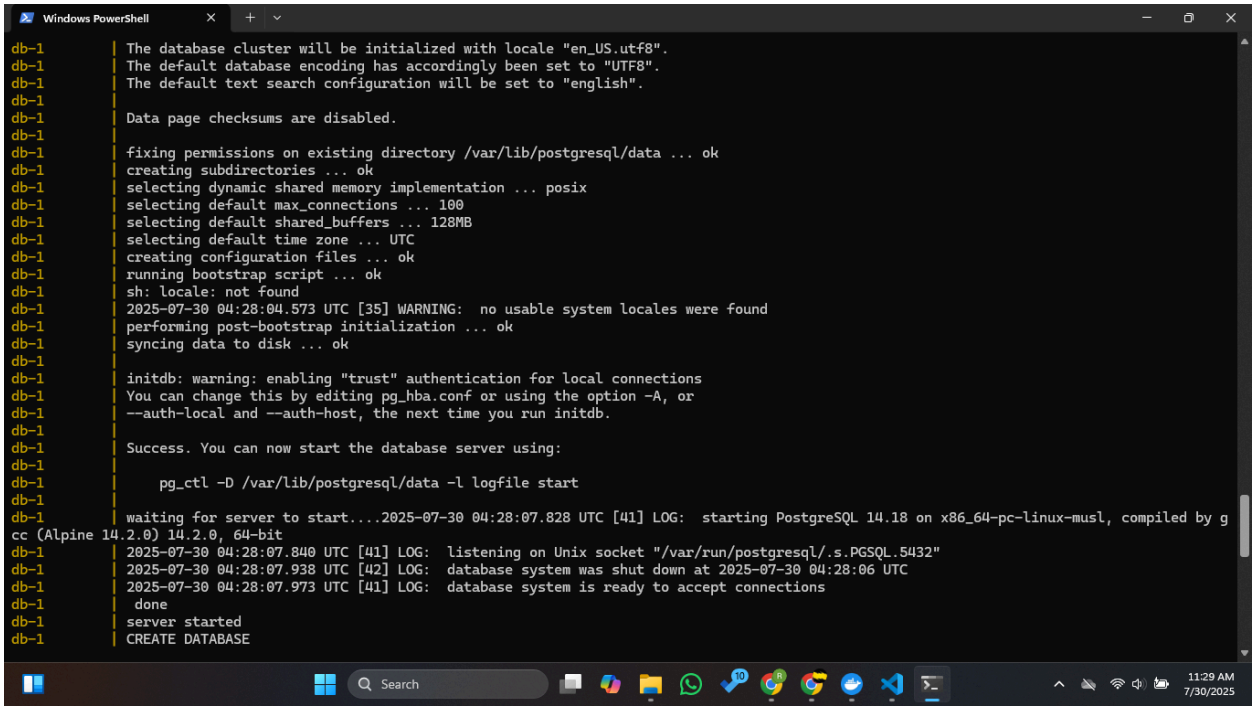
1. Pembuatan docker-compose.yml: Sebuah file docker-compose.yml dibuat di direktori utama.
    - Fungsi: File ini bertindak sebagai "sutradara" atau orkestrator lokal. Ia mendefinisikan layanan-layanan yang membentuk aplikasi (db, backend, frontend), mengatur bagaimana mereka terhubung melalui jaringan, dan mengelola konfigurasi seperti *port mapping* dan *environment variables*.
    - Analisis: Docker Compose dipilih untuk menyederhanakan proses pengembangan lokal secara drastis. Tanpa Docker Compose, developer harus menjalankan dan mengonfigurasi tiga kontainer secara manual, yang sangat tidak efisien.
2. Menjalankan dan Menguji: Dengan satu perintah, docker-compose up --build, keseluruhan tumpukan teknologi (database, backend, frontend) dibangun dan dijalankan. Aplikasi kemudian diuji melalui browser di http://localhost:3000 untuk memvalidasi bahwa semua *tier* berkomunikasi dengan benar sesuai desain arsitektur. Seluruh proses *troubleshooting* yang kita lalui adalah bagian dari tahap pengujian ini.

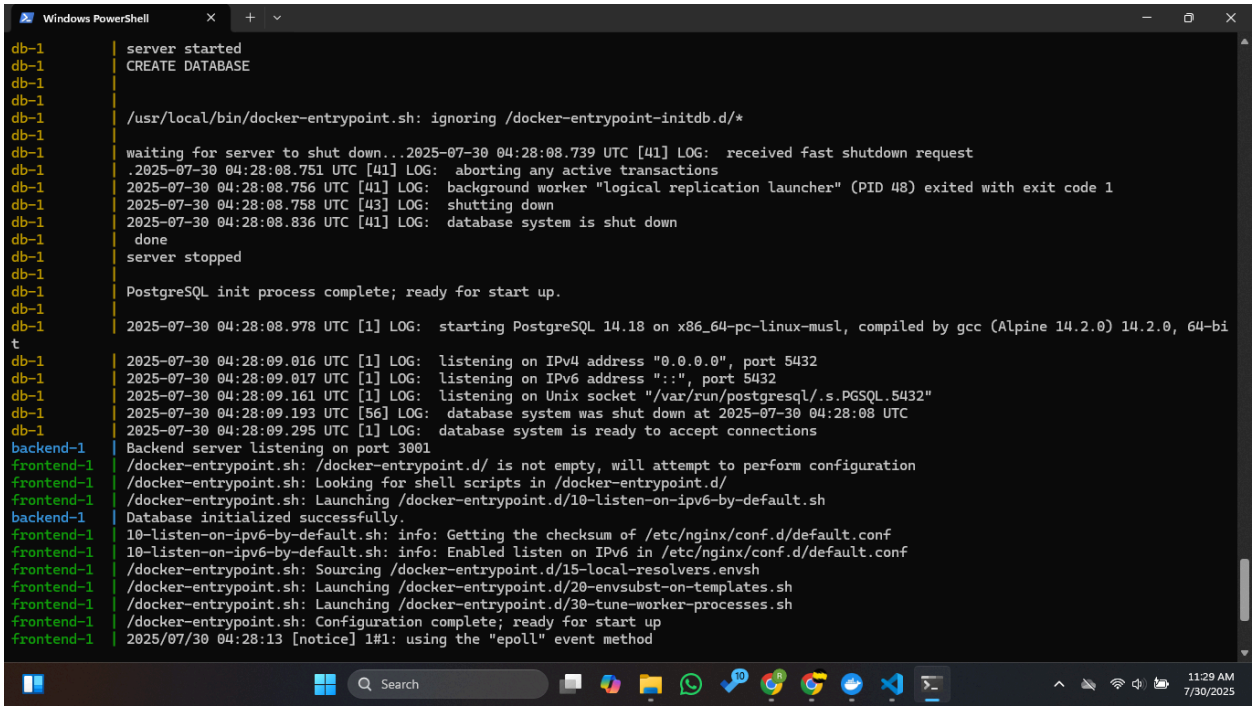
Dokumentasi Pengerjaan

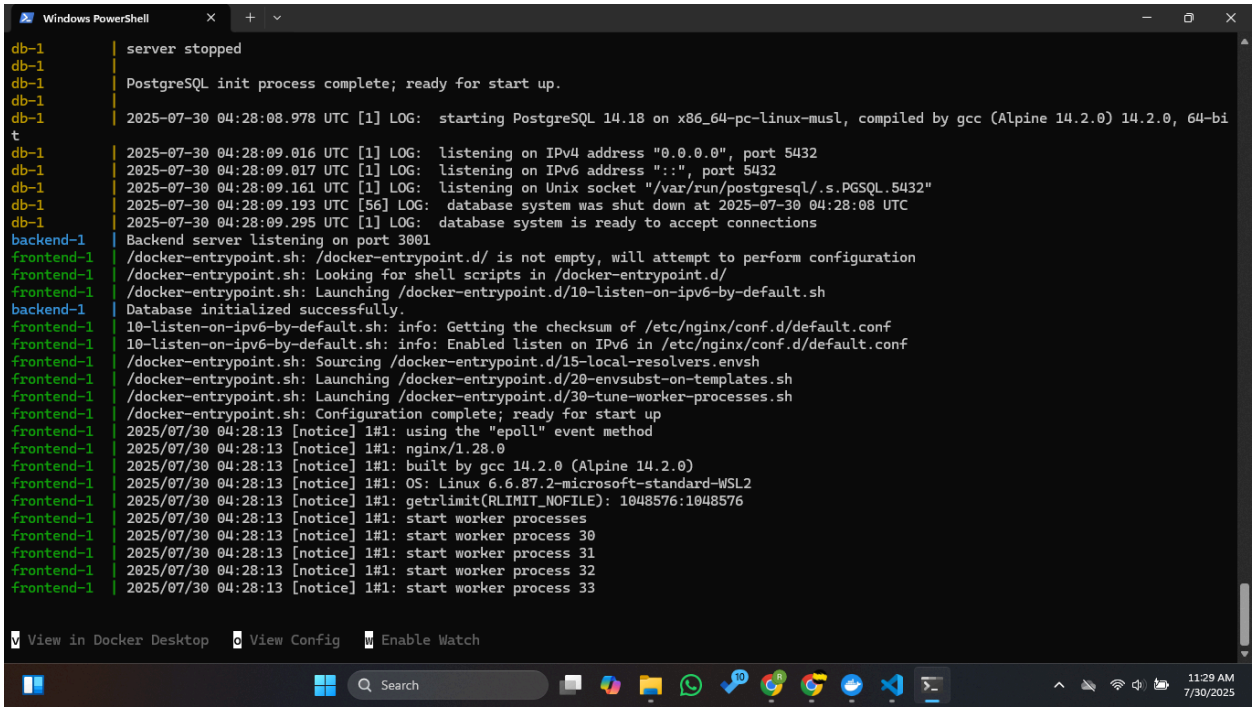
Dibawah ini merupakan dokumentasi dari proses membangun dan menjalankan aplikasi menggunakan docker yang saya kerjakan pada terminal laptop saya. Setelah semuanya berhasil, kemudian saya mengujinya langsung pada web dengan alamat http://localhost:3000 di laptop saya untuk membuktikan apakah sesuai dengan spesifikasi yang diinginkan atau tidak.

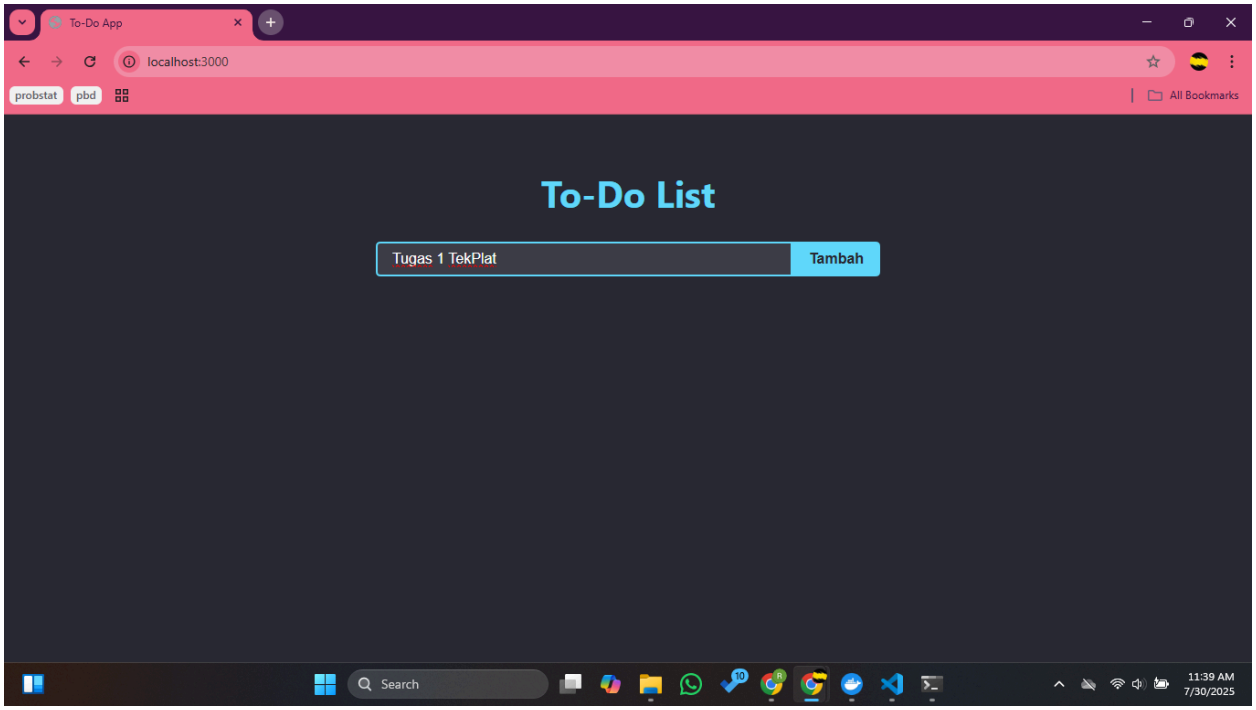


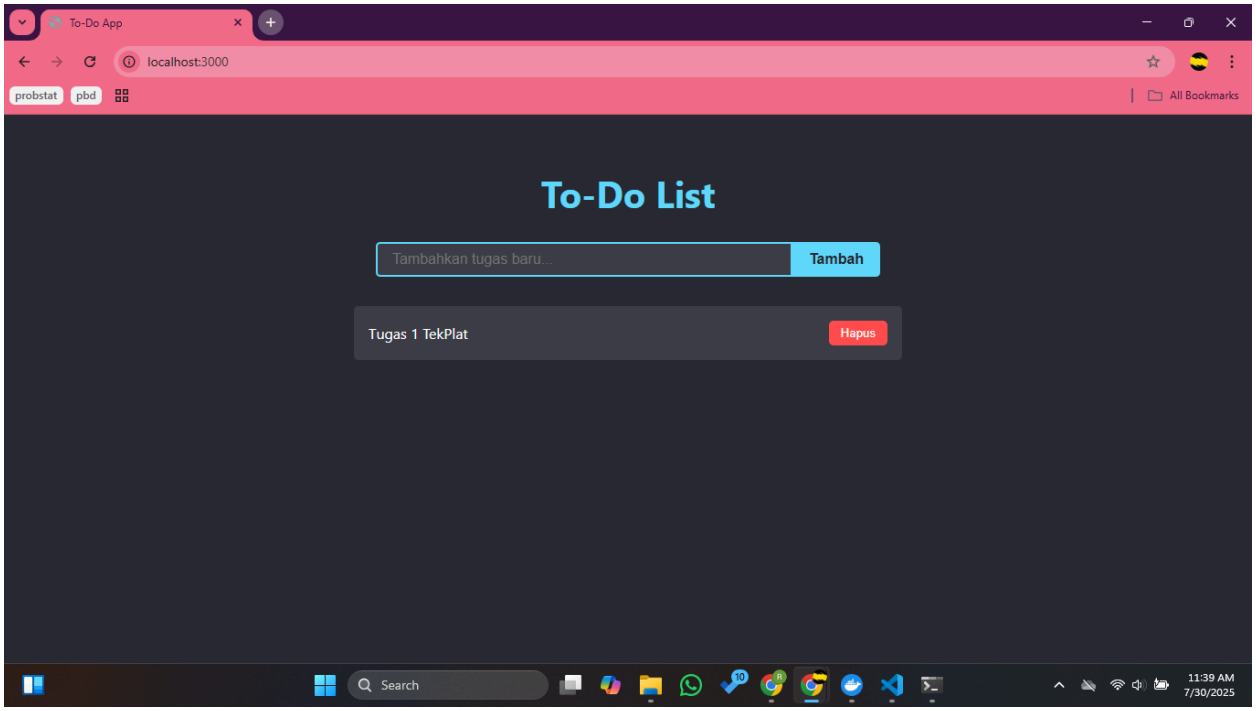


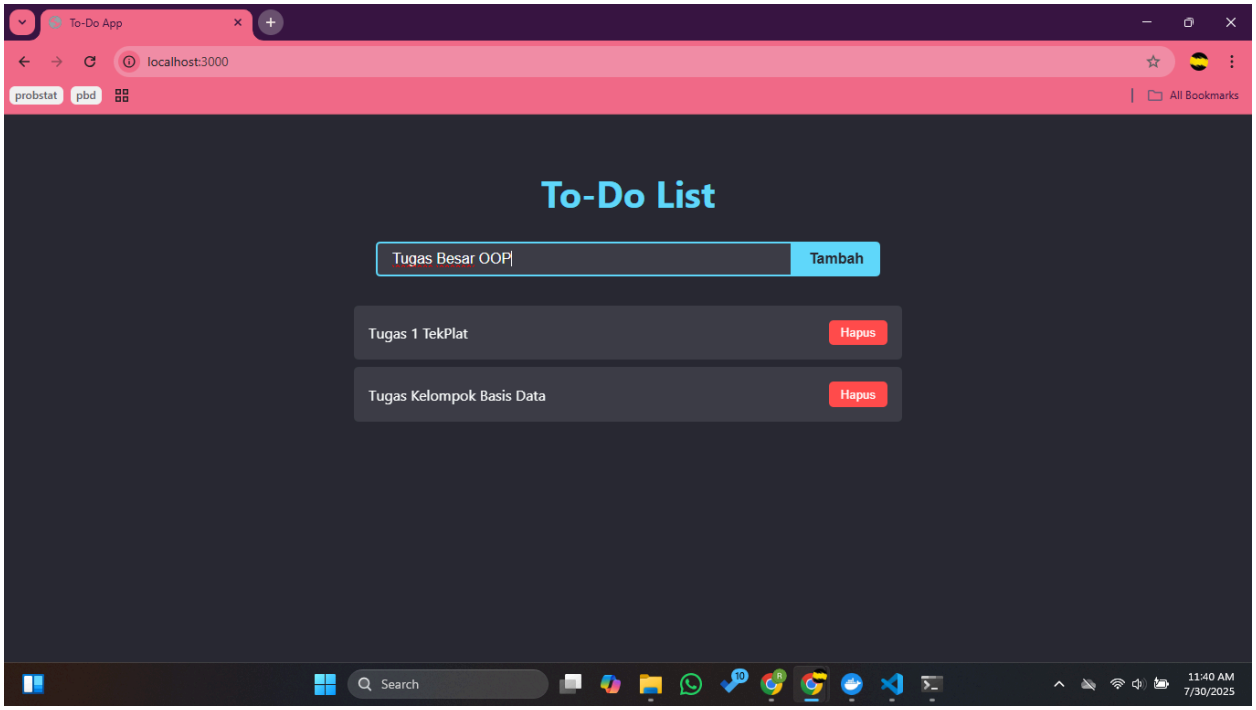

Windows PowerShell
db-1 | The database cluster will be initialized with locale "en_US.utf8".
db-1 | The default database encoding has accordingly been set to "UTF8".
db-1 | The default text search configuration will be set to "english".
db-1 |
db-1 | Data page checksums are disabled.
db-1 |
db-1 | fixing permissions on existing directory /var/lib/postgresql/data ... ok
db-1 | creating subdirectories ... ok
db-1 | selecting dynamic shared memory implementation ... posix
db-1 | selecting default max_connections ... 100
db-1 | selecting default shared_buffers ... 128MB
db-1 | selecting default time zone ... UTC
db-1 | creating configuration files ... ok
db-1 | running bootstrap script ... ok
db-1 | sh: locale: not found
db-1 | 2025-07-30 04:28:04.573 UTC [35] WARNING: no usable system locales were found
db-1 | performing post-bootstrap initialization ... ok
db-1 | syncing data to disk ... ok
db-1 |
db-1 | initdb: warning: enabling "trust" authentication for local connections
db-1 | You can change this by editing pg_hba.conf or using the option -A, or
db-1 | --auth-local and --auth-host, the next time you run initdb.
db-1 |
db-1 | Success. You can now start the database server using:
db-1 |
db-1 | pg_ctl -D /var/lib/postgresql/data -l logfile start
db-1 |
db-1 | waiting for server to start....2025-07-30 04:28:07.828 UTC [41] LOG: starting PostgreSQL 14.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1 | 2025-07-30 04:28:07.840 UTC [41] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1 | 2025-07-30 04:28:07.938 UTC [42] LOG: database system was shut down at 2025-07-30 04:28:06 UTC
db-1 | 2025-07-30 04:28:07.973 UTC [41] LOG: database system is ready to accept connections
db-1 | done
db-1 | server started
db-1 | CREATE DATABASE
Search
11:29 AM
7/30/2025

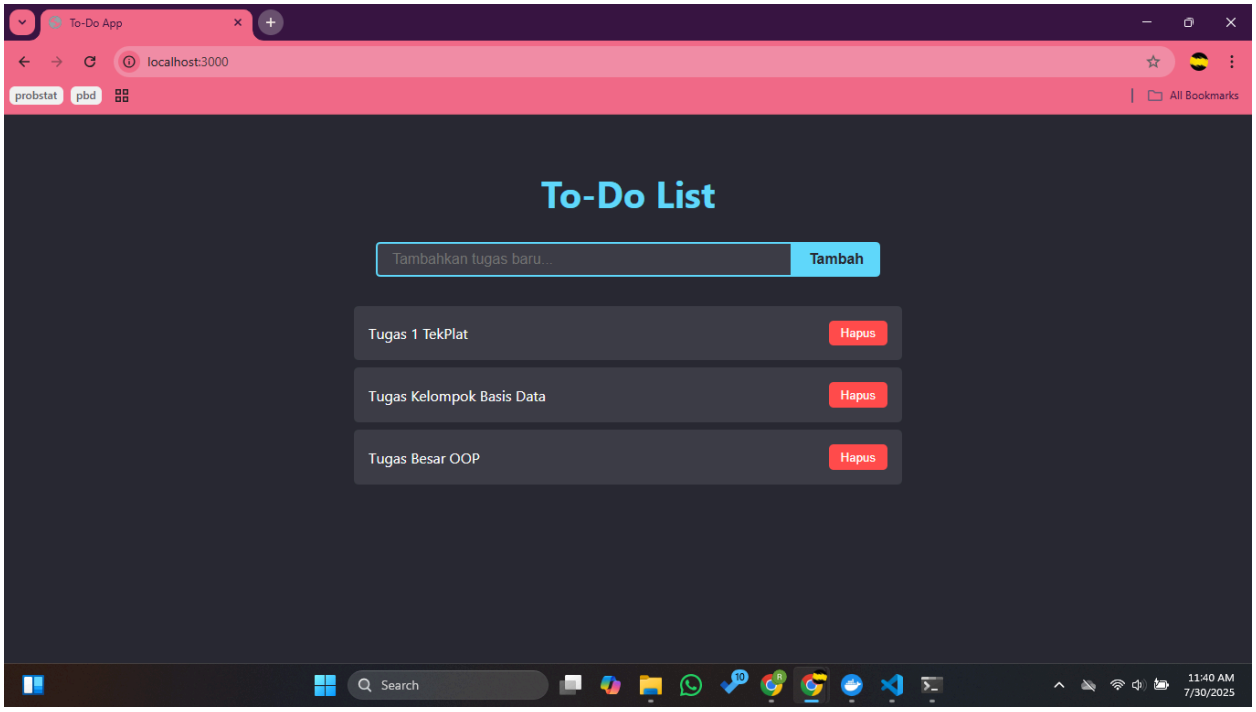

Windows PowerShell
db-1 | server started
db-1 | CREATE DATABASE
db-1 |
db-1 |
db-1 | /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*
db-1 |
db-1 | waiting for server to shut down...2025-07-30 04:28:08.739 UTC [41] LOG: received fast shutdown request
db-1 | .2025-07-30 04:28:08.751 UTC [41] LOG: aborting any active transactions
db-1 | 2025-07-30 04:28:08.756 UTC [41] LOG: background worker "logical replication launcher" (PID 48) exited with exit code 1
db-1 | 2025-07-30 04:28:08.758 UTC [43] LOG: shutting down
db-1 | 2025-07-30 04:28:08.836 UTC [41] LOG: database system is shut down
db-1 | done
db-1 | server stopped
db-1 |
db-1 | PostgreSQL init process complete; ready for start up.
db-1 |
db-1 | 2025-07-30 04:28:08.978 UTC [1] LOG: starting PostgreSQL 14.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bi
t
db-1 | 2025-07-30 04:28:09.016 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1 | 2025-07-30 04:28:09.017 UTC [1] LOG: listening on IPv6 address "::", port 5432
db-1 | 2025-07-30 04:28:09.161 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1 | 2025-07-30 04:28:09.193 UTC [56] LOG: database system was shut down at 2025-07-30 04:28:08 UTC
db-1 | 2025-07-30 04:28:09.295 UTC [1] LOG: database system is ready to accept connections
backend-1 | Backend server listening on port 3001
frontend-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
frontend-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
backend-1 | Database initialized successfully.
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
frontend-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
frontend-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: using the "epoll" event method
Search
11:29 AM
7/30/2025

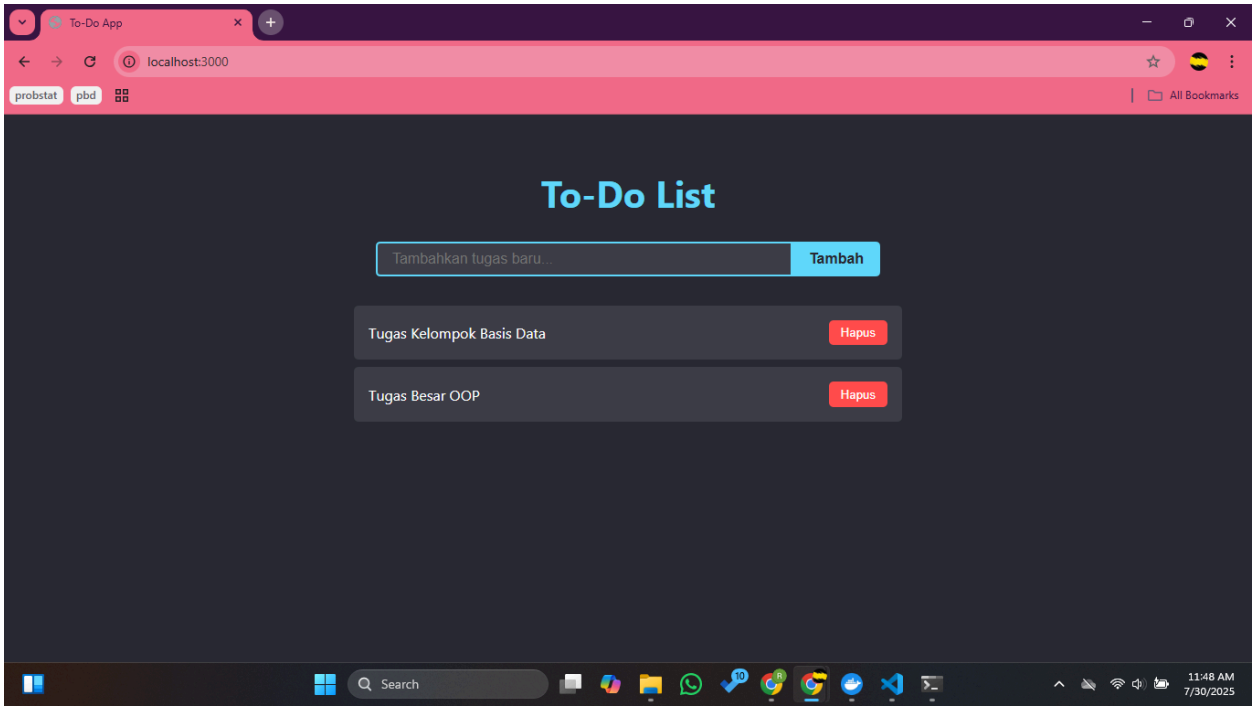

Windows PowerShell
db-1 | server stopped
db-1 |
db-1 | PostgreSQL init process complete; ready for start up.
db-1 |
db-1 | 2025-07-30 04:28:08.978 UTC [1] LOG: starting PostgreSQL 14.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bi
t
db-1 | 2025-07-30 04:28:09.016 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1 | 2025-07-30 04:28:09.017 UTC [1] LOG: listening on IPv6 address "::", port 5432
db-1 | 2025-07-30 04:28:09.161 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1 | 2025-07-30 04:28:09.193 UTC [56] LOG: database system was shut down at 2025-07-30 04:28:08 UTC
db-1 | 2025-07-30 04:28:09.295 UTC [1] LOG: database system is ready to accept connections
backend-1 | Backend server listening on port 3001
frontend-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
frontend-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
backend-1 | Database initialized successfully.
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
frontend-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
frontend-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: using the "epoll" event method
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: nginx/1.28.0
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: OS: Linux 6.6.87.2-microsoft-standard-WSL2
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: start worker processes
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: start worker process 30
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: start worker process 31
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: start worker process 32
frontend-1 | 2025/07/30 04:28:13 [notice] 1#1: start worker process 33
View in Docker Desktop
View Config
Enable Watch
Search
11:29 AM
7/30/2025

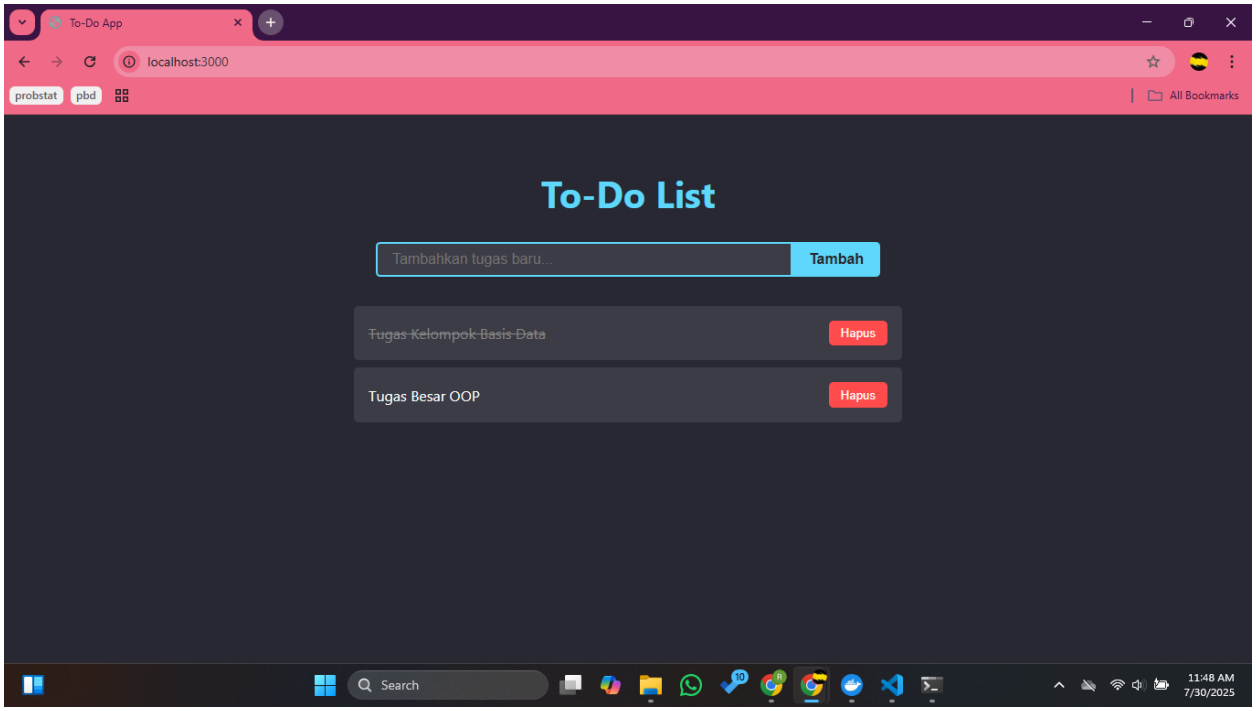

To-Do App
localhost:3000
probstat
pbd
All Bookmarks
To-Do List
Tugas 1 TekPlat
Tambah
Search
11:39 AM
7/30/2025


To-Do App
localhost:3000
probstat
pbd
All Bookmarks
To-Do List
Tambahkan tugas baru...
Tambah
Tugas 1 TekPlat
Hapus
Search
11:39 AM
7/30/2025


To-Do App
localhost:3000
probstat
pbd
All Bookmarks
To-Do List
Tugas Besar OOP
Tambah
Tugas 1 TekPlat
Hapus
Tugas Kelompok Basis Data
Hapus
Search
11:40 AM
7/30/2025


To-Do App
localhost:3000
probstat
pbd
All Bookmarks
To-Do List
Tambahkan tugas baru...
Tambah
Tugas 1 TekPlat
Hapus
Tugas Kelompok Basis Data
Hapus
Tugas Besar OOP
Hapus
Search
11:40 AM
7/30/2025





Berdasarkan pengujian, dibuktikan bahwa To-Do App dapat menambahkan tugas baru (dengan mengetikkan tugas baru lalu klik tombol “Tambah”), menghapus tugas yang sudah ada (dengan klik tombol “Hapus” yang ada di sebelah kanan setiap tugas yang ada) , dan menandai tugas yang sudah dikerjakan (tugas yang dicoret dalam tampilan frontend, dengan klik bagian tugas).